

OS MID SEM NOV 1

Question :

Dear PALETI KRISHNASAI,

Question: Develop a Multiprocessing Version of Radix Sort algorithm and compare it with the performance (execution time) of bubble and insertion sort algorithms. The demonstration should display the passes of the respective sorting strategies. The comparison is against the Multiprocessing Radix sort with sequential versions of Bubble and Insertion Sort Algorithm. Ensure that the testing explores large sized arrays, random distribution of elements. As an addon it would be preferred to have a data creator code which initializes an array of user required size randomly given some boundary conditions.

Answer :

Radix Sort (Multiprocessing)

Radix Sort sorts the elements by initially grouping the individual digits of the same place value. The idea of Radix Sort is to do digit by digit sort *starting from least significant digit(LSD) to the most significant digit(MSD)*, according to their increasing/decreasing order.

Algorithm

```
Radix_sort (list, n)
shift = 1
for loop = 1 to keysize do
    for entry = 1 to n do
        bucketnumber = (list[entry].key / shift) mod 10
        append (bucket[bucketnumber], list[entry])
    list = combinebuckets()
    shift = shift * 10
```

In a multiprocessing setup , the total number of passes have been split up , where half the passes are done in the child process and the trailing half are done in the parent process.

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <string.h>
#include <time.h>
#include <sys/time.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <sys/stat.h>
#include <unistd.h>
#include <fcntl.h>
#define MAX 100000

int get_max(int a[], int n)
{
    int max = a[0];
    for (int i=1;i<n;i++)
        if(a[i]>max)
            max = a[i];
    return max;
}

int main()
{
    srand(time(0));
    struct timeval start, stop;
    double secs = 0;

    int output_fds = open("./output.txt",O_WRONLY | O_APPEND);
    dup2(output_fds,STDOUT_FILENO);
    int size,array[MAX];
    printf("Enter the size of the list : \n");
    scanf("%d",&size);
    for(int i=0;i<size;i++)
```

```
{
    array[i] = rand()%(size+1);
}
/* for (i = 0; i < 100; i++)
{
    printf("%d --->  %d\n",i+1,array[i]);
}
*/
gettimeofday(&start, NULL);
int bucket[size][size], bucket_count[size];
int i, j, k, r, NOP = 0, divisor = 1, lar, pass;
lar = get_max(array, size);

int pid = vfork();
if(pid==0)
{

    while(lar > 0)
    {
        NOP++;
        lar /= 10;

    }
    for(pass=0;pass<NOP/2;pass++)
    {
        for(i=0;i<size;i++)
        {
            bucket_count[i] = 0;
        }
        for(i =0;i<size;i++)
        {
            r = (array[i]/divisor)%10;
            bucket[r][bucket_count[r]] = array[i];
            bucket_count[r] += 1;

        }
        i=0;
        for(k=0;k<size;k++)
        {
            for (j=0;j<bucket_count[k];j++)
```

```
        {
            array[i] = bucket[k][j];
            i++;
        }
    }
    divisor *= 10;
    printf("PASS %d : \n",pass+1);
    for(i =0;i<size;i++)
    {
        printf("%d ",array[i]);
    }
    printf("\n");
}
exit(0);
}
if(pid>0)
{
    wait(NULL);
    for (pass = NOP/2; pass < NOP; pass++)
    {
        for (i = 0; i < size; i++)
        {
            bucket_count[i] = 0;
        }
        for (i = 0; i < size; i++)
        {
            r = (array[i] / divisor) % 10;
            bucket[r][bucket_count[r]] = array[i];
            bucket_count[r] += 1;
        }
        i = 0;
        for (k = 0; k < size; k++)
        {
            for (j = 0; j < bucket_count[k]; j++)
            {
                array[i] = bucket[k][j];
                i++;
            }
        }
        divisor *= 10;
    }
}
```

```
        printf("\n");
        printf("PASS %d : \n", pass + 1);
        printf("\n");
        for (i = 0; i < size; i++)
        {
            printf("%d ", array[i]);
        }
        printf("\n");
    }
}

gettimeofday(&stop, NULL);
secs = (double)(stop.tv_usec - start.tv_usec) / 1000000 +
(double)(stop.tv_sec - start.tv_sec);

printf("\n");
printf("Sorted list : \n");
printf("\n");
for (i=0;i<size;i++)
printf("%d ",array[i]);
printf("\n");
printf("Execution time:  %f\n",secs);

return 0;
}
```

Insertion and Bubble Sort :

Classic Serial implementation of bubble and insertion sort .

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <string.h>
#include <time.h>
#include <sys/time.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <sys/stat.h>
#include <unistd.h>
#include <fcntl.h>
#define MAX 100000

void insertionSort(int arr[], int n)
{
    int i, key, j;
    for (i = 1; i < n; i++)
    {
        key = arr[i];
        j = i - 1;

        /* Move elements of arr[0..i-1], that are
           greater than key, to one position ahead
           of their current position */
        while (j >= 0 && arr[j] > key)
        {
            arr[j + 1] = arr[j];
            j = j - 1;
        }
        arr[j + 1] = key;
        for (int k = 0; k < n; ++k)
            printf(" %d", arr[k]);
        printf("\n");
    }
}
```

```
void printArray1(int arr[], int n)
{
    int i;
    for (i = 0; i < n; i++)
        printf("%d ", arr[i]);
    printf("\n");
}

void swap(int *xp, int *yp)
{
    int temp = *xp;
    *xp = *yp;
    *yp = temp;
}

// A function to implement bubble sort
void bubbleSort(int arr[], int n)
{
    int i, j, pass = 0;
    for (i = 0; i < n - 1; i++)
    {
        // Last i elements are already in place
        for (j = 0; j < n - i - 1; j++)
            if (arr[j] > arr[j + 1])
                swap(&arr[j], &arr[j + 1]);
        //pass++;
        //printf(" After pass %d:", pass);
        for (int k = 0; k < n; ++k)
            printf(" %d", arr[k]);
        printf("\n");
    }
}

/* Function to print an array */
void printArray2(int arr[], int size)
{
    int i;
    for (i = 0; i < size; i++)
        printf("%d ", arr[i]);
    printf("\n");
}
```

```
int main()
{
    srand(time(0));
    struct timeval start, stop;
    double secs = 0;
    //int output_fds = open("./output.txt", O_WRONLY);
    //dup2(output_fds, STDOUT_FILENO);
    int size, array1[MAX], array2[MAX];
    printf("Enter the size of the list : \n");
    scanf("%d", &size);
    for (int i = 0; i < size; i++)
    {
        array1[i] = rand() % size+1;
        array2[i] = array1[i];
    }
    printf("\nArray:");
    for (int i = 0; i < size; i++)
    {
        printf(" %d", array1[i]);
    }
    printf("\n");
    printf("-----\n\n");

    gettimeofday(&start, NULL);

    insertionSort(array1, size);
    printf("\n");
    printArray1(array1, size);
    printf("-----\n\n");
    printf("\n");
    bubbleSort(array2, size);
    printf("\n");
    printArray2(array2, size);
    gettimeofday(&stop, NULL);
    secs = (double)(stop.tv_usec - start.tv_usec) / 1000000 +
(double)(stop.tv_sec - start.tv_sec);

    printf("\n");
    printf("Execution time:  %f\n", secs);
```



```
return 0;  
}
```

Output :

Will print in a file.

dummy.txt = terminal output of radix sort copied to a file.

dummy1.txt = terminal output of bubble and insertion sort copied to a file.

output.txt = output to a file using dup2() system call.

output1.txt = output of bubble and insertion sort to a file using dup2() system call. // need to comment out the respective lines in the program to get this functionality.

Execution times have been compared and it can be noted that the multiprocessing radix sort is slightly slower than the serial implementation of bubble and insertion sort for input sizes of 100 or lesser.

But when input size goes to 1000 , then we can see significant difference in execution time .(pic shown below)

FEW screenshots are shown below.(codes and required files will be attached in a zip file)

```
csdscsorc.x Question.txt RadixSort.x output.txt x dummy.txt Insertion_Bubble.x dummy1.txt output1.txt
MIDSEM > output.txt
You, a few seconds ago | 1 author (You)
1 Enter the size of the list : You, 9 minutes ago · midsem
2 PASS 1 :
3 280 850 550 490 450 890 860 420 610 890 550 950 1000 130 110 720 980 40 770 880 110 410 710 820 110 810 290 670 600 100 130 350 910 300 270 660 810 60 880 70 680
4 PASS 2 :
5 1000 600 100 380 880 380 1000 600 400 0 400 200 300 0 500 101 901 1 501 301 801 201 501 701 1 1 801 801 701 401 401 801 201 702 102 802 902 602 503 403 903 303 80
6
7 PASS 3 :
8
9 1000 1000 0 0 1 1 1 4 4 6 8 9 10 10 11 11 13 13 15 18 20 21 27 31 31 32 32 33 34 34 34 35 37 38 38 38 39 39 40 41 41 42 44 44 44 48 51 51 52 53 53 53 54 55 5
10
11 PASS 4 :
12
13 0 0 1 1 1 4 4 6 8 9 10 10 11 11 13 13 15 18 20 21 21 27 31 31 32 32 33 34 34 34 35 37 38 38 38 39 39 40 41 41 42 44 44 48 51 51 52 53 53 53 54 55 57 58 58
14
15 Sorted list :
16
17 0 0 1 1 1 4 4 4 6 8 9 10 10 11 11 13 13 15 18 20 21 21 27 31 31 32 32 33 34 34 34 35 37 38 38 38 39 39 40 41 41 42 44 44 44 48 51 51 52 53 53 53 54 55 55 57 58 58
18 Execution time : 0.014423
19 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 3 3 3 3 3 3 3 3 3 3 3 4 4 4 4 4 4 4 4 5 5 5 5 5 5 5 5 5 6 6 6 6 6 6 6 6 6 6 6 7 7 7 7 7 7 7 7 7 8 8 8 8 8 8 8 8 9 9 9 9 9 9 9
20 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 3 3 3 3 3 3 3 3 3 3 3 4 4 4 4 4 4 4 4 4 5 5 5 5 5 5 5 5 5 6 6 6 6 6 6 6 6 6 6 6 6 6 6 7 7 7 7 7 7 7 7 7
21 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
22 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
23
24 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
25 Execution time : 0.001396
26
27
TERMINAL PROBLEMS OUTPUT DEBUG CONSOLE
1: bash
844 846 847 848 849 850 850 851 851 856 857 857 859 860 862 862 862 863 863 864 865 865 865 865 866 866 866 866 868 869 870 872 872 873 874 874 875 875 876 881 885 885 885 886 887 8
87 888 888 888 889 889 895 895 895 898 901 901 902 902 903 903 904 905 905 905 906 907 907 907 909 909 910 911 912 913 915 915 916 918 919 921 923 925 926 926 927 927 928 929 932 93
2 933 933 934 935 935 936 936 937 937 938 939 941 942 944 945 945 946 946 946 947 948 949 951 953 953 955 958 960 960 961 962 962 962 963 963 963 965 965 970 970 970 973 974 975 975 978
978 978 979 979 979 983 984 986 988 990 990 993 993 995 995 997 998 998 1000 1000 1000
Execution time : 13.700239
paletigpaletil:~/OS_LAB/MIDSEM$ gcc RadixSort.c
paletigpaletil:~/OS_LAB/MIDSEM$ ./a.out
1000
paletigpaletil:~/OS_LAB/MIDSEM$
```

Radix sort : Execution time: 0.001396

```

689 993 694 695 696 696 697 701 701 702 703 704 706 707 708 708 709 711 712 712 713 715 716 717 717 717 718 719 720 721 722 723 723 724 724 724 725 726 726 727 727 728
31 733 734 735 736 736 737 738 739 740 742 744 744 745 745 746 746 748 748 750 752 753 755 756 761 763 763 767 768 771 771 775 776 777 778 780 780 783 783 785 786 788
7 788 790 790 790 791 793 794 795 795 797 797 797 797 801 802 802 805 806 806 809 810 811 811 812 812 814 814 817 818 820 820 820 820 822 822 824 826 827 828 830 830 834 839 841
844 846 847 848 849 850 850 851 851 856 857 857 859 860 862 862 863 863 864 865 865 865 866 866 868 869 870 872 872 873 874 874 875 876 881 885 885 885 885 886 887
887 888 888 888 889 889 895 895 895 898 901 901 902 902 903 903 904 905 905 906 907 907 909 909 910 911 912 913 915 915 916 918 919 921 923 925 926 926 927 928 929 932 9
32 933 934 935 935 936 936 937 937 938 939 941 942 944 945 945 946 946 947 948 949 951 953 953 955 956 960 961 962 962 963 963 963 965 970 970 970 973 974 975 975 978
978 978 979 979 979 980 983 984 986 988 990 990 993 993 995 995 997 998 998 1000 1000 1000
3 3 4 4 5 5 6 6 8 8 9 9 10 10 11 11 11 11 12 14 15 18 21 22 22 22 24 25 26 28 28 29 29 29 35 36 36 38 38 38 44 47 48 48 48 48 49 49 51 51 53 54 55 57 58 58 59 59 62 64 64 64 6
6 67 67 68 68 69 73 74 74 74 75 76 77 79 80 80 81 81 82 82 81 82 82 82 88 88 92 94 96 97 99 100 100 101 102 103 103 105 105 105 105 106 106 107 107 109 112 112 111 112
113 113 115 115 116 116 119 121 121 122 123 125 126 126 127 127 128 129 130 131 131 132 133 133 134 134 136 136 138 140 140 143 143 145 145 146 147 148 149 149 149 151 1
51 152 152 152 154 156 157 158 158 160 162 162 165 166 166 166 168 168 173 174 175 176 176 177 179 180 181 182 182 184 185 186 187 187 188 189 189 189 192 193 193 194 196 198 19
9 200 202 203 207 210 215 214 215 218 219 219 224 225 225 228 228 228 228 229 230 230 231 231 232 234 234 237 237 237 238 239 241 242 242 243 244 245 246 250 250 250 250
252 252 252 253 253 255 255 255 256 256 257 258 258 260 260 260 262 262 264 264 264 265 265 266 268 270 278 271 271 272 274 274 276 277 278 278 278 278 282 282 283 283 284 285
286 286 286 286 286 286 286 286 286 286 286 286 286 286 286 286 286 286 286 286 286 286 286 286 286 286 286 286 286 286 286 286 286 286 286 286 286 286 286 286 286 286 286 286
35 336 337 339 340 341 342 342 343 344 344 346 347 348 348 350 352 352 353 355 356 357 357 358 358 359 359 361 361 364 364 364 366 367 367 367 367 367 367 367 367 367 367 367 367 367
379 379 380 382 384 386 388 387 392 392 393 394 397 398 399 399 400 402 403 403 404 404 405 405 405 406 406 406 407 407 409 411 414 414 419 419 422 422 422 422 423 423 426 427
428 429 430 431 431 432 432 437 439 440 441 443 445 446 446 448 450 451 452 453 456 456 459 459 461 461 462 463 464 464 466 466 467 468 468 470 471 474 474 477 477 477 477
478 479 479 479 480 480 481 481 481 483 485 485 486 486 487 488 488 488 490 491 491 491 492 492 494 494 496 498 498 498 499 499 501 501 501 502 503 503 503 504 506 506 506 507 507 5
509 510 516 516 518 518 518 518 519 521 521 522 523 524 525 525 526 528 528 531 532 533 534 537 539 539 541 543 544 545 545 546 547 550 551 551 552 553 553 554 555 555
7 557 558 561 561 565 566 566 567 570 571 571 572 572 573 576 577 578 578 580 580 581 582 588 589 589 594 594 600 603 604 605 605 607 608 608 609 609 610 611 612 613 614 615
616 616 616 616 616 616 616 616 616 616 616 616 616 616 616 616 616 616 616 616 616 616 616 616 616 616 616 616 616 616 616 616 616 616 616 616 616 616 616 616 616 616 616 616
640 650 650 651 652 652 653 653 654 654 658 660 660 663 663 666 667 668 668 669 671 672 672 672 676 677 677 677 677 677 677 677 677 677 677 677 677 677 677 677 677 677 677 677 677 677
693 694 695 696 696 697 701 701 702 703 704 704 706 707 708 708 709 711 712 712 713 715 716 716 717 717 717 718 719 720 721 722 723 723 724 724 724 726 726 726 727 727 73
1 733 734 735 735 736 737 738 739 740 742 744 744 745 745 746 746 748 748 750 752 753 755 756 756 761 763 763 767 768 771 771 775 776 777 778 780 780 783 783 785 786 787
788 790 790 790 79
```

Insertion and bubble sort : Execution time: 13.700239

Printing of passes for a small input size for readability (insertion and bubble sort)

```
paleti@paletil:~/OS_LAB/MIDSEM$ ./Insertion_Bubble
```

```
Enter the size of the list :
```

```
10
```

```
Array: 10 10 4 3 10 5 10 3 8 10
```

```
-----
```

```
10 10 4 3 10 5 10 3 8 10
4 10 10 3 10 5 10 3 8 10
3 4 10 10 10 5 10 3 8 10
3 4 10 10 10 5 10 3 8 10
3 4 5 10 10 10 10 3 8 10
3 4 5 10 10 10 10 3 8 10
3 3 4 5 10 10 10 10 8 10
3 3 4 5 8 10 10 10 10 10
3 3 4 5 8 10 10 10 10 10
```

```
3 3 4 5 8 10 10 10 10 10
```

```
-----
```

```
10 4 3 10 5 10 3 8 10 10
4 3 10 5 10 3 8 10 10 10
3 4 5 10 3 8 10 10 10 10
3 4 5 3 8 10 10 10 10 10
3 4 3 5 8 10 10 10 10 10
3 3 4 5 8 10 10 10 10 10
3 3 4 5 8 10 10 10 10 10
3 3 4 5 8 10 10 10 10 10
3 3 4 5 8 10 10 10 10 10
```

```
3 3 4 5 8 10 10 10 10 10
```

```
Execution time: 0.000175
```

```
RadixSort.c  output.txt X  dummy.txt  Insertion_Bubble.c  ...  output1.txt X ...
MIDSEM > output.txt
4 PASS 2 :
5 0 600 700 400 600 1000 900 1 301 701 901 801 101 101 101 401 101 501 901 1
6
7 PASS 3 :
8
9 0 1000 1 1 1 2 4 4 5 6 7 7 8 9 10 10 11 12 15 16 17 17 18 19 20 20 21
10
11 PASS 4 :
12
13 0 1 1 1 2 4 4 5 6 7 7 8 9 10 10 11 12 15 16 17 17 18 19 20 20 21 22 2
14
15 Sorted list :
16
17 0 1 1 1 2 4 4 5 6 7 7 8 9 10 10 11 12 15 16 17 17 18 19 20 20 21 22 2
18 Execution time: 0.016169
19

MIDSEM > output1.txt
1104 993 ---> 187
1105 994 ---> 355
1106 995 ---> 154
1107 996 ---> 901
1108 997 ---> 52
1109 998 ---> 9
1110 999 ---> 668
1111 1000 ---> 453
1112
1113 0 1 2 2 3 4 6 7 8 9 10 11 11 12 12 12 13 14 14 14 19 23 23 25 25 28 29 29
1114
1115
1116 0 1 2 2 3 4 6 7 8 9 10 11 11 12 12 13 14 14 14 19 23 23 25 25 28 29 29
1117
1118 Execution time: 0.014619
1119

paleti@paleti:~/OS_LAB/MIDSEM$ ./a.out
100
paleti@paleti:~/OS_LAB/MIDSEM$ ./a.out
100
paleti@paleti:~/OS_LAB/MIDSEM$ gcc RadixSort.c
paleti@paleti:~/OS_LAB/MIDSEM$ ./a.out
1000
paleti@paleti:~/OS_LAB/MIDSEM$ make Insertion_Bubble
make: 'Insertion_Bubble' is up to date.
paleti@paleti:~/OS_LAB/MIDSEM$ ./Insertion_Bubble
1000
paleti@paleti:~/OS_LAB/MIDSEM$
```

```
File Edit Selection View Go Run Terminal Help
n.txt  RadixSort.c  output.txt X  dummy.txt  Insertion_Bubble.c  ...  output1.txt X ...
MIDSEM > output.txt
1 Enter the size of the list :
2 PASS 1 :
3 0 0 40 90 90 90 90 0 81 11 11 71 61 11 31 61 11 31 32 52 12 22 72 22 82 2
4
5 PASS 2 :
6
7 0 0 0 2 3 3 3 5 7 7 7 8 9 11 11 11 11 12 15 15 16 16 16 19 19 22 22 2
8
9 Sorted list :
10
11 0 0 0 2 3 3 3 5 7 7 7 8 9 11 11 11 11 12 15 15 16 16 16 19 19 22 22 2
12 Execution time: 0.000608
13

MIDSEM > output1.txt
97 95 ---> 75
98 96 ---> 34
99 97 ---> 46
100 98 ---> 100
101 99 ---> 54
102 100 ---> 40
103
104 0 1 2 2 4 5 7 9 11 12 13 13 13 13 14 18 20 21 22 24 24 25 25 27 28 29 32 3
105
106
107 0 1 2 2 4 5 7 9 11 12 13 13 13 13 14 18 20 21 22 24 24 25 25 27 28 29 32 3
108
109 Execution time: 0.000174
110
```

```
paleti@paleti:~/OS_LAB/MIDSEM$ gcc RadixSort.c
paleti@paleti:~/OS_LAB/MIDSEM$ ./a.out
Enter the size of the list :
100
PASS 1 :
100 30 80 10 50 0 40 10 50 10 30 50 91 21 51 81 51 1 71 51 72 22 52 52 92 12 62 92 92 92 42 2 63 73 63 63 93 43 13 94 74 64 14 44 74 14 74 44 54 74 75 45 15 5 55 35 36 66 96
6 26 6 6 26 16 86 97 27 47 27 77 7 87 77 47 47 87 47 7 88 28 38 48 68 48 58 98 98 28 99 99 19 19 59 39 69 99 19

PASS 2 :

100 0 1 2 5 6 6 7 7 10 10 10 12 13 14 14 15 16 19 19 19 21 22 26 26 27 27 28 28 30 30 35 36 38 39 40 42 43 44 44 45 47 47 47 47 48 48 50 50 50 51 51 51 52 52 54 55 58 59 62 63 63
63 64 66 66 68 69 71 72 73 74 74 74 74 75 77 77 80 81 86 87 87 88 91 92 92 92 92 93 94 96 96 97 98 98 99 99 99

PASS 3 :

0 1 2 5 6 6 7 7 10 10 10 12 13 14 14 15 16 19 19 19 21 22 26 26 27 27 28 28 30 30 35 36 38 39 40 42 43 44 44 45 47 47 47 47 48 48 50 50 50 51 51 51 52 52 54 55 58 59 62 63 63
64 66 66 68 69 71 72 73 74 74 74 74 75 77 77 80 81 86 87 87 88 91 92 92 92 92 93 94 96 96 97 98 98 99 99 99 100

Sorted list :

0 1 2 5 6 6 7 7 10 10 10 12 13 14 14 15 16 19 19 19 21 22 26 26 27 27 28 28 30 30 35 36 38 39 40 42 43 44 44 45 47 47 47 47 48 48 50 50 50 51 51 51 52 52 54 55 58 59 62 63 63
64 66 66 68 69 71 72 73 74 74 74 74 75 77 77 80 81 86 87 87 88 91 92 92 92 92 93 94 96 96 97 98 98 99 99 99 100
Execution time: 0.002326
```